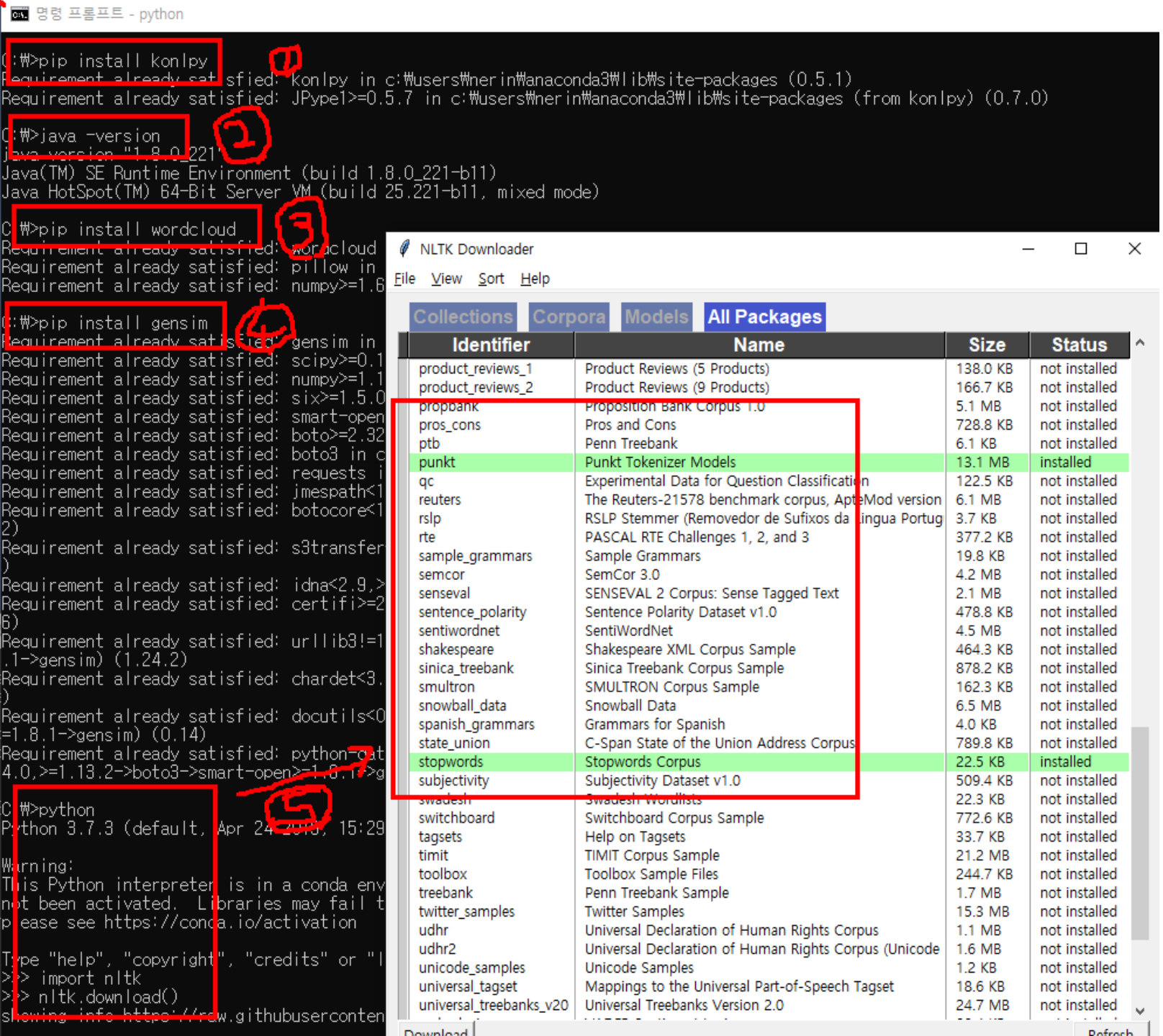
8장 자연어 처리 시작하기

1. 한글 자연어 처리 기초 – KoNLPy 및 필요 모듈의 설치
2. 한글 자연어 처리 기초
3. 워드 클라우드
4. 육아휴직 관련 법안에 대한 분석
5. Naive Bayes Classifier의 이해 - 영문
6. Naive Bayes Classifier의 이해 - 한글
7. 문장의 유사도 측정하기
8. 여자친구 선물 고르기
9. 소감

"**코엔엘파이**"라고 읽는다.



1. KoNLPy 및 필요 모듈의 설치

2. 한글 자연어 처리 기초

In [1]:
```
#8-2 한글 자연어 처리 기초
```

In [2]:
```
# 8장에서 사용할 모듈들 설치
# 1. cmd >>> pip install konlpy
# 파이썬으로 한국어 정보처리를 할 수있게 하는 패키지
# 2. JDK install
# 3. JDK 환경변수 설정
# JAVA_HOME = "C:₩Program Files"₩"Java"₩"jdk-version"
# 4. cmd >>> python
#             import nltk
#             nltk.download()
# All Package 탭에서 stopwords와 punkt를 설치
# 5. cmd >>> pip install wordcloud
# 6. cmd >>> pip install genism
```

In [3]:
```
# 꼬꼬마 모델 엔진
from konlpy.tag import Kkma
kkma = Kkma()
```

In [4]:
```
# 문장을 분석했습니다
kkma.sentences('한국어 분석을 시작합니다 재미있어요~~')
```

Out[4]:
```
['한국어 분석을 시작합니다', '재미있어요~~']
```

In [5]:
```
# 명사를 분석했습니다
kkma.nouns('한국어 분석을 시작합니다 재미있어요~~')
```

Out[5]:
```
['한국어', '분석']
```

In [6]:
```
# 최소한의 의미 단위인 형태소를 분석합니다
kkma.pos('한국어 분석을 시작합니다 재미있어요~~')
```

Out[6]:
```
[('한국어', 'NNG'),
 ('분석', 'NNG'),
 ('을', 'JKO'),
 ('시작하', 'VV'),
 ('ㅂ니다', 'EFN'),
 ('재미있', 'VA'),
 ('어요', 'EFN'),
 ('~~', 'SW')]
```

In [7]:
```
# 한나눔 모델 엔진
from konlpy.tag import Hannanum
hannanum = Hannanum()
```

In [8]:
```
# 문장을 분석했습니다
hannanum.nouns('한국어 분석 시작합니다 우유는 맛없다 집에 가고싶어')
```

In [9]:
```
# 명사를 분석합니다
hannanum.morphs('한국어 분석 시작합니다 우유는 맛없다 집에 가고싶어')
```

Out[9]:
```
['한국어',
 '분석',
 '시작',
 '하',
 'ㅂ니다',
 '우유',
 '는',
 '맛없',
 '다',
 '집',
 '에',
 '가',
 '고',
 '싶',
 '어']
```

In [10]:
```
# 형태소를 분석합니다
hannanum.pos('한국어 분석 시작합니다 우유는 맛없다 집에 가고싶어')
```

Out[10]:
```
[('한국어', 'N'),
 ('분석', 'N'),
 ('시작', 'N'),
 ('하', 'X'),
 ('ㅂ니다', 'E'),
 ('우유', 'N'),
 ('는', 'J'),
 ('맛없', 'P'),
 ('다', 'E'),
 ('집', 'N'),
 ('에', 'J'),
 ('가', 'P'),
 ('고', 'E'),
 ('싶', 'P'),
 ('어', 'E')]
```

In [11]:
```
# 트위터 모델 엔진
# 버전 경고
from konlpy.tag import Twitter
t = Twitter()
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitt
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
```

In [12]:
```
# 문장을 분석했습니다
t.nouns('한국어 분석 시작합니다 우유는 맛없다 집에 가고싶어')
```

Out[12]:
```
['한국어', '분석', '시작', '우유', '집']
```

In [13]:
```
# 명사를 분석합니다
t.morphs('한국어 분석 시작합니다 우유는 맛없다 집에 가고싶어')
```

Out[13]:
```
['한국어', '분석', '시작', '합니다', '우유', '는', '맛없다', '집', '에', '가고싶어']
```

In [14]:
```
# 형태소를 분석합니다
t.pos('한국어 분석 시작합니다 우유는 맛없다 집에 가고싶어')
```

Out[14]:
```
[('한국어', 'Noun'),
 ('분석', 'Noun'),
 ('시작', 'Noun'),
 ('합니다', 'Verb'),
 ('우유', 'Noun'),
 ('는', 'Josa'),
 ('맛없다', 'Adjective'),
 ('집', 'Noun'),
 ('에', 'Josa'),
 ('가고싶어', 'Verb')]
```

3. 워드 클라우드

In [15]:
```
# 8-3 워드 클라우드
```

In [16]:
```
# 자주 나타나는 단어를 크게 보여줌으로써 직관적으로 텍스트를
# 알려주는 wordcloud와 빈도수를 계산할 때 해당 단어를 제외시키는
# stopwords(불용어)를 임포트합니다
# 파이썬에서 이미지를 처리하고 핸들링하기 위해 Pillow 패키지를 사용합니다.
# PIL 패키지에서 이미지를 표현하는 Image 클래스를 임포트합니다
from wordcloud import WordCloud, STOPWORDS

import numpy as np
from PIL import Image
```

In [17]:
```
# text 변수에 이상한 나라의 앨리스 소설 영문 버전 alice.txt를
# alice_mask 변수에 앨리스 그림파일을 읽었습니다
# 앨리스 소설에서 자주 나타나는 said를 제거했습니다
text = open('../data/09. alice.txt').read()
alice_mask = np.array(Image.open('../data/09. alice_mask.png'))

stopwords = set(STOPWORDS)
stopwords.add("said")
```

In [18]:
```
# 한글 폰트 설정
import matplotlib.pyplot as plt
import platform

path = "c:/Windows/Fonts/malgun.ttf"
from matplotlib import font_manager, rc
if platform.system() == 'Darwin':
    rc('font', family='AppleGothic')
elif platform.system() == 'Windows':
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
else:
    print('Unknown system... sorry~~~~')

%matplotlib inline
```

In [19]:
```
# imshow cmap
# 데이터 수치를 색으로 바꾸는 칼라맵 지원
# imshow interpolation
# 명령의 시각화를 돕기위해 다양한 2차원 인터폴레이션 지원
# https://datascienceschool.net/view-notebook/6e71dbff254542d9b0a054a7c98b34ec/
# axis('off')로 가로 세로 label을 제거했습니다
# https://frhyme.github.io/python-lib/plt_axis_off/
plt.figure(figsize=(4,4))
plt.imshow(alice_mask, cmap=plt.cm.winter, interpolation='bilinear')
plt.axis('off')
plt.show()
```

In [20]:
```
# 워드 클라우드는 문서 자체에서 단어를 카운트하므로
# 앨리스 소설에서 가장 많이 등장하는 단어를 카운트합니다
# Alice가 가장 많이 등장했음을 알 수 있습니다.
wc = WordCloud(background_color='white', max_words=2000, mask=alice_mask,
               stopwords = stopwords)
wc = wc.generate(text)
wc.words_
```

Out[20]:
```
{'Alice': 1.0,
 'little': 0.2958904109589041,
 'one': 0.2602739726027397,
 'know': 0.2465753424657534,
 'went': 0.2273972602739726,
 'thing': 0.2191780821917808,
 'time': 0.21095890410958903,
 'Queen': 0.20821917808219179,
 'see': 0.18356164383561643,
 'King': 0.17534246575342466,
```

In [21]:
```
# 위의 코드 결과를 앨리스 그림에 겹쳐서 출력합니다
plt.figure(figsize=(4,4))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.show()
```

In [22]:
```
# 스타워즈의 A New Hope 텍스트와 그림파일을 읽었습니다.
text = open('../data/09. a_new_hope.txt').read()

text = text.replace('HAN', 'Han')
text = text.replace("LUKE'S", 'Luke')

mask = np.array(Image.open('../data/09. stormtrooper_mask.png'))
```

In [23]:
```
# 텍스트에서 int와 ext 단어를 제거합니다.
stopwords = set(STOPWORDS)
stopwords.add("int")
stopwords.add("ext")
```

In [24]:
```
# 워드 클라우드의 최대 단어의 개수를 1000개로 제한하는 등,
# 워드 클라우드를 준비합니다.
wc = WordCloud(max_words=1000, mask=mask, stopwords=stopwords,
               margin=10, random_state=1).generate(text)

default_colors = wc.to_array()
```

In [25]:
```
# 색상을 전체적으로 회색으로 채우는 함수를 작성합니다.
# grey_color_func
import random
def grey_color_func(word, font_size, position, orientation,
                    random_state=None, **kwargs):
    return 'hsl(0, 0%%, %d%%)' % random.randint(60,100)
```

In [26]:
```
# 워드 클라우드를 사용해서 그림을 그립니다.
plt.figure(figsize=(6,6))
plt.imshow(wc.recolor(color_func=grey_color_func, random_state=3),
           interpolation='bilinear')
plt.axis('off')
plt.show()
```

4. 육아휴직 관련 법안에 대한 분석

In [27]:
```
# 8-4 육아휴직 관련 법안에 대한 분석
```

In [28]:
```
# 텍스트에서 의미있는 정보의 분석과 추출을
# 도와주는 자연어 처리 패키지 nltk 임포트
import nltk
```

In [29]:
```
# KoNLPy의 내부 문서의
# 육아휴직 관련 법안 제 1809890호를 읽었습니다.
from konlpy.corpus import kobill
files_ko = kobill.fileids()
doc_ko = kobill.open('1809890.txt').read()
```

In [30]:
```
# 공무원 한정으로 육아휴직대상이 되는
# 아이의 나이를 만 6세부터 만 8세로
# 확장한다는 내용
doc_ko
```

Out[30]:
```
'지방공무원법 일부개정법률안₩n₩n(정의화의원 대표발의 )₩n₩n 의 ?
화.이명수.김을동 ₩n₩n이사철.여상규.안규백₩n₩n황영철.박영아.김정훈
우에도 부모의 따뜻한 사랑과 보살핌이 필요₩n₩n한 나이이나, 현재
만 6세 이하로 되어 있어 초등학교 저학년인 ₩n₩n자녀를 돌보기 위
하시키는 문제로 이어질 수 있을 것임.₩n₩n  따라서 육아휴직이 가
호).₩n₩n- 1 -₩n₩n₩x0c법률  제        호₩n₩n지방공무원법 일부개
중 "만 6세 이하의 초등학교 취학 전 자녀를" 을 "만 ₩n₩n8세 이
다.₩n₩n부      칙₩n₩n이 법은 공포한 날부터 시행한다.₩n₩n- 3 -₩
(생  략)₩n₩n제63조(휴직) ① (현행과 같음)₩n₩n  ② 공무원이 다음
로 휴₩n₩n-------------------------₩n₩n직을 원하면 임용권자는
```

In [31]:
```
# Twitter 분석기로 육아휴직 관련 법안을 명사분석 했습니다.
# 버전에러
from konlpy.tag import Twitter; t = Twitter()
tokens_ko = t.nouns(doc_ko)
tokens_ko
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩konlpy₩tag₩_okt.py:1
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
```

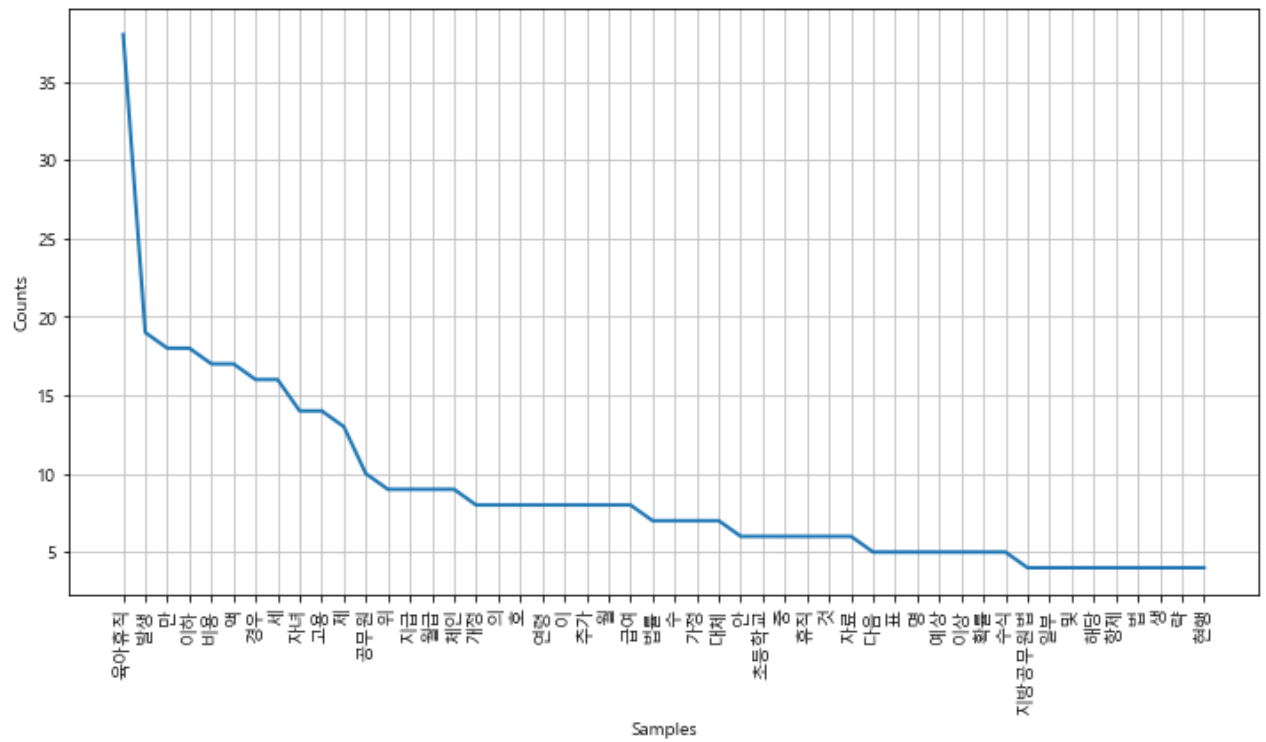
Out[31]:
```
['지방공무원법',
 '일부',
 '개정',
 '법률',
 '안',
 '정의화',
 '의원',
 '대표',
 '발의',
 '의',
 '안',
 '번',
 '호',
 '발의',
 '연월일',
 '발',
```

In [32]:
```
# nltk로 육아휴직 관련 법안의 단어를 수집하고
# 수집한 단어의 횟수와 고유한 횟수를 확인했습니다.
ko = nltk.Text(tokens_ko, name='대한민국 국회 의안 제 1809890호')

print(len(ko.tokens))       # returns number of tokens (document length)
print(len(set(ko.tokens)))  # returns number of unique tokens
ko.vocab()                  # returns frequency distribution
```

```
735
250
```

Out[32]:
```
FreqDist({'육아휴직': 38, '발생': 19, '만': 18, '이하': 18, '비용': 17, '액': 17, '경우': 16, '세': 16, '자녀': 14, '고용': 14, ...})
```

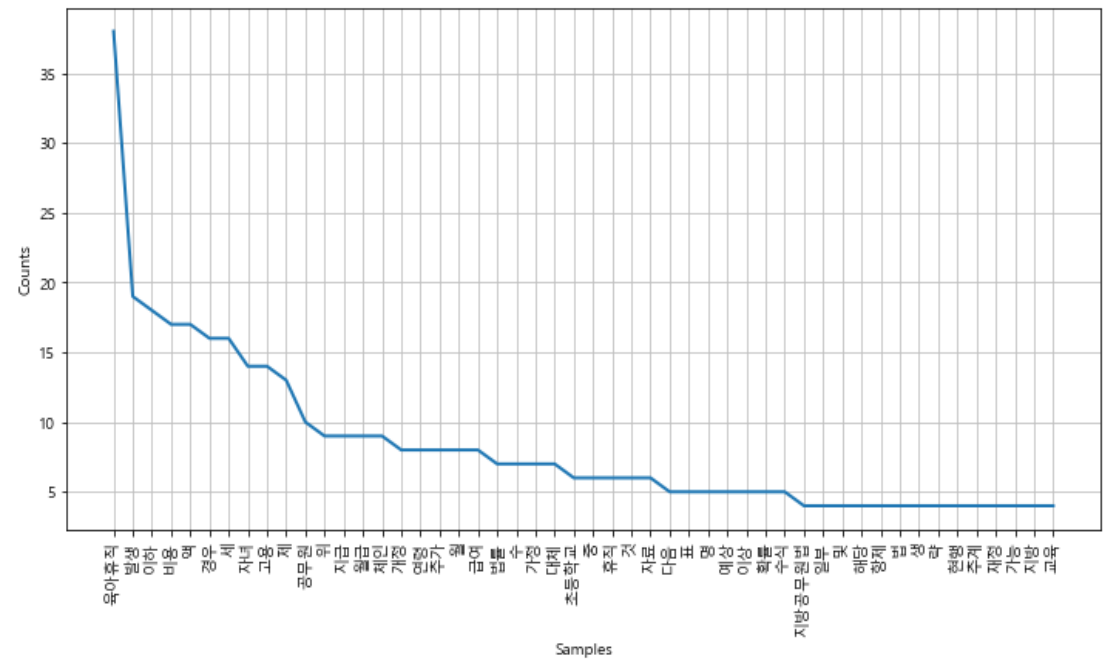
In [33]:
```
# 단어가 많이 등장한 순으로 그래프를 그렸습니다.
# 당연히 육아휴직이란 단어가 많이 등장하고,
# 의미없는 단어들도 많이 출력되었습니다.
plt.figure(figsize=(12,6))
ko.plot(50)
plt.show()
```



In [34]:
```
# 의미없는 단어를 제거했습니다.
# 한글은 영어와 달리 stopwords를 지정하기 쉽지 않아서
# 제거할 때 case-by-case를 지정해서 삭제했습니다.
stop_words = ['.', '(', ')', ',', "'", '%', '-', 'X', ').', '×','의','자','에','안','번',
                    '호','을','이','다','만','로','가','를']

ko = [each_word for each_word in ko if each_word not in stop_words]
ko
```

Out [34]:
```
['지방공무원법',
 '일부',
 '개정',
 '법률',
 '정의화',
 '의원',
 '대표',
 '발의',
 '발의',
 '연월일',
 '발',
 '정의화',
 '이명수',
 '김을동',
 '이사철',
 '여상규',
 '안규백',
 '황영철',
 '박영아',
```

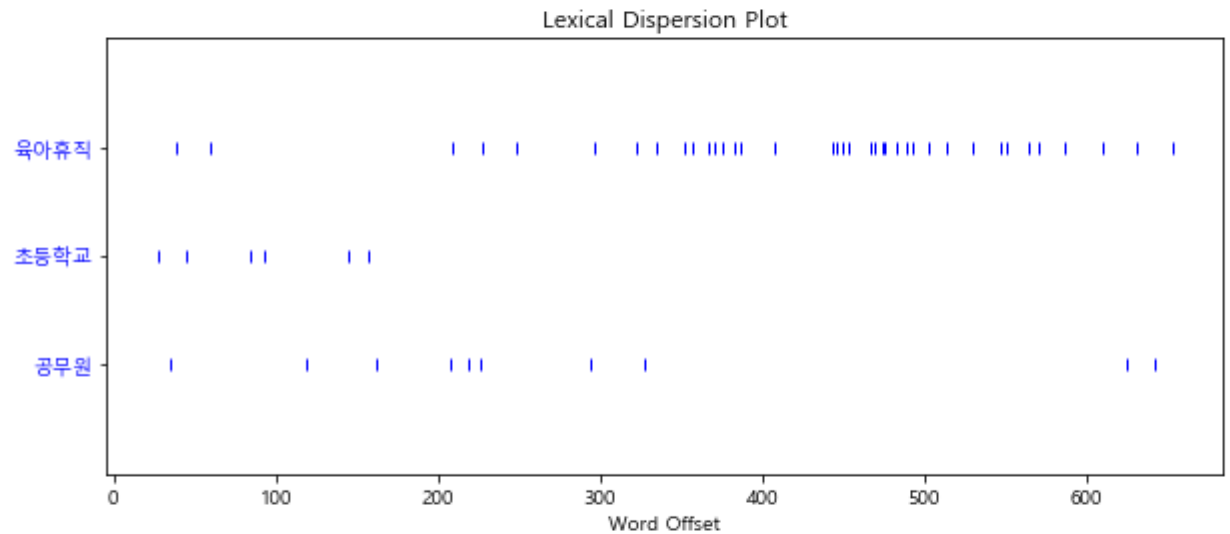
In [35]:
```
# 의미없는 단어를 삭제하고 그래프를 다시 그렸습니다.
ko = nltk.Text(ko, name='대한민국 국회 의안 제 1809890호')

plt.figure(figsize=(12,6))
ko.plot(50)     # Plot sorted frequency of top 50 tokens
plt.show()
```



In [36]:
```
# 초등학교라는 단어가 몇 번
# 등장했는지 출력합니다
ko.count('초등학교')
```

Out [36]: 6

In [37]:
```
# 지정한 단어의 문서 내 위치를 개략적으로 출력합니다.
plt.figure(figsize=(10,4))
ko.dispersion_plot(['육아휴직', '초등학교', '공무원'])
```



In [38]:
```
# 지정한 단어의 주변 단어를 확인합니다.
ko.concordance('초등학교')
```

```
Displaying 6 of 6 matches:
 안규백 황영철 박영아 김정훈 김학송 의원 인 제안 이유 및 내용 초등학교 저학년 경우 부모 사랑 필요 나이 현재 공무원 자녀 양육 위 육아
 나이 현재 공무원 자녀 양육 위 육아휴직 수 자녀 나이 세 이하 초등학교 저학년 자녀 위 해당 부모님 일자리 곧 출산 의욕 저하 문제 수
일부 개정 법률 지방공무원법 일부 다음 개정 제 항제 중 세 이하 초등학교 취학 전 자녀 세 이하 취학 중인 경우 초등학교 학년 이하 말 자
항제 중 세 이하 초등학교 취학 전 자녀 세 이하 취학 중인 경우 초등학교 학년 이하 말 자녀 부 칙 법 공포 날 시행 신 구조 문대비 표
 수 다만 제 경우 대통령령 정 사정 직 명 생 략 현행 세 이하 초등학교 취 세 이하 취학 중인 경우 학 전 자녀 양육 위 초등학교 학년
 이하 초등학교 취 세 이하 취학 중인 경우 학 전 자녀 양육 위 초등학교 학년 이하 여 여자 공무원 말 자녀 임신 출산 때 생 략 생 략
```

In [39]:
```
# ko.collocations() error!
# -> print('; '.join(ko.collocation_list())) 수정
# 문서 내에서 어떤 단어들이 연속된 단어로 사용되었는지 출력합니다.
print('; '.join(ko.collocation_list()))
```

초등학교 저학년; 근로자 육아휴직; 육아휴직 대상자; 공무원 육아휴직

In [40]:
```
# 지금까지 학습한 워드 클라우드를 출력합니다.
# malgun.ttf : 윈도우 10에서 적용
# AppleGothic.ttf : 맥OS에서 적용
# 출력하기 전에 윈도우 한글 폰트로 바꿔주었습니다.
data = ko.vocab().most_common(150)

# for win : font_path='c:/Windows/Fonts/AppleGothic.ttf'
wordcloud = WordCloud(font_path='/Library/Fonts/malgun.ttf',
                      relative_scaling = 0.2,
                      background_color='white',
                      ).generate_from_frequencies(dict(data))
plt.figure(figsize=(8,4))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

5. Naive Bayes Classifier의 이해 - 영문

In [41]:
```
# 8-5 Naive Bayes Classifier의 이해 - 영문
# 지도학습의 한 종류이면서 두 사건을 서로 독립이라
# 가정하고 각각의 조건부 확률을 구한다는 개념
# [i like you는 긍정, you like me는 부정
# 두 문장만 봤을 때 i가 등장하면 긍정이고,
# like는 판단하지 못한다.]
```

In [42]:
```
# 텍스트에서 의미있는 정보의 분석과 추출을
# 도와주는 자연어 처리 패키지 nltk 임포트
from nltk.tokenize import word_tokenize
import nltk
```

In [43]:
```
# 연습용 데이터
train = [('i like you', 'pos'),
         ('i hate you', 'neg'),
         ('you like me', 'neg'),
         ('i like her', 'pos')]
```

In [44]:
```
# 연습용 데이터에서 사용된 전체 단어를 출력합니다
# 출력된 단어를 말뭉치라고 정의하겠습니다
all_words = set(word.lower()
                for sentence in train
                for word in word_tokenize(sentence[0]))
all_words
```

Out[44]:
```
{'hate', 'her', 'i', 'like', 'me', 'you'}
```

In [45]:
```
# 말뭉치를 기준으로
# 연습용 데이터에 속한
# 단어인지 아닌지 기록합니다
# 예를들어, 연습용 데이터의 첫 문장인 i like you의 경우
# 말뭉치 단어를 기준으로 해당 단어가 있는지 없는지를 기록합니다
t = [({word: (word in word_tokenize(x[0]))
       for word in all_words}, x[1])
       for x in train]
t
```

Out[45]:
```
[({'hate': False,
   'her': False,
   'i': True,
   'like': True,
   'you': True,
   'me': False},
  'pos'),
 ({'hate': True,
   'her': False,
   'i': True,
   'like': False,
   'you': True,
   'me': False},
  'neg'),
 ({'hate': False,
   'her': False,
   'i': False,
   'like': True,
   'you': True,
```

In [46]:
```
# 연습용 데이터에 붙어있는 긍정/부정 태그를 이용해서 분류한 결과
# hate라는 단어가 없을 때 긍정일 비율이 1.7:1 이고
# like라는 단어가 있을 때 긍정일 확률이 1.7:1 이다
classifier = nltk.NaiveBayesClassifier.train(t)
classifier.show_most_informative_features()
```

```
Most Informative Features
                      me = False           pos : neg    =      1.7 : 1.0
                    like = True            pos : neg    =      1.7 : 1.0
                       i = True            pos : neg    =      1.7 : 1.0
                     you = True            neg : pos    =      1.7 : 1.0
                    hate = False           pos : neg    =      1.7 : 1.0
                     her = False           neg : pos    =      1.7 : 1.0
```

In [47]:
```
# 새로 만든 테스트 문장이 분류기를 거치도록 합니다
test_sentence = 'i like bigdata'
test_sent_features = {word.lower():
                      (word in word_tokenize(test_sentence.lower()))
                      for word in all_words}
test_sent_features
```

Out[47]:
```
{'hate': False,
 'her': False,
 'i': True,
 'like': True,
 'you': False,
 'me': False}
```

In [48]:
```
# 긍정의 결과가 나옵니다
# like를 hate로 바꾸면 neg가 출력됩니다
classifier.classify(test_sent_features)
```

Out[48]:
```
'pos'
```

6. Naive Bayes Classifier의 이해 - 한글

In [49]:
```
# 8-6 Naive Bayes Classifier
```

In [50]:
```
# Twitter 모델 엔진을 임포트 했습니다.
# 8-5절의 내용을 한글로 적용합니다.
from konlpy.tag import Twitter
pos_tagger = Twitter()
```

In [51]:
```
# 우선 형태소를 분석하지 않고 적용했습니다.
# 연습용 데이터를 생성합니다
train = [('메리가 좋아', 'pos'),
         ('고양이도 좋아', 'pos'),
         ('난 수업이 지루해', 'neg'),
         ('메리는 이쁜 고양이야', 'pos'),
         ('난 마치고 메리랑 놀거야', 'pos')]
```

In [52]:
```
# 연습용 데이터로 말뭉치를 생성했습니다.
# 고양이도, 고양이야 / 메리가, 메리는, 메리랑이
# 다른 단어로 잡혔습니다.
all_words = set(word.lower()
                for sentence in train
                for word in word_tokenize(sentence[0]))
all_words
```

Out [52]:
```
{'고양이도',
 '고양이야',
 '난',
 '놀거야',
 '마치고',
 '메리가',
 '메리는',
 '메리랑',
 '수업이',
 '이쁜',
 '좋아',
 '지루해'}
```

In [53]:
```
# 말뭉치를 기준으로 연습용 데이터에 속한
# 단어인지 아닌지 기록합니다
t = [({word: (word in word_tokenize(x[0]))
       for word in all_words}, x[1])
       for x in train]
t
```

Out[53]:
```
[({'지루해': False,
   '메리는': False,
   '이쁜': False,
   '수업이': False,
   '난': False,
   '메리가': True,
   '마치고': False,
   '고양이야': False,
   '좋아': True,
   '고양이도': False,
   '놀거야': False,
   '메리랑': False},
  'pos'),
 ({'지루해': False,
   '메리는': False,
   '이쁜': False,
   '수업이': False,
   '난': False,
   '메리가': False,
```

In [54]:
```
# 연습용 데이터에 붙어있는 긍정/부정 태그를 이용해서 분류했습니다.
classifier = nltk.NaiveBayesClassifier.train(t)
classifier.show_most_informative_features()
```

```
Most Informative Features
                난 = True              neg : pos    =      2.5 : 1.0
               좋아 = False             neg : pos    =      1.5 : 1.0
              놀거야 = False             neg : pos    =      1.1 : 1.0
              메리는 = False             neg : pos    =      1.1 : 1.0
              마치고 = False             neg : pos    =      1.1 : 1.0
              메리랑 = False             neg : pos    =      1.1 : 1.0
             고양이야 = False             neg : pos    =      1.1 : 1.0
              메리가 = False             neg : pos    =      1.1 : 1.0
               이쁜 = False             neg : pos    =      1.1 : 1.0
             고양이도 = False             neg : pos    =      1.1 : 1.0
```

In [55]:
```
# 새로 만든 긍정의 결과가 나올것 같은 테스트 문장이
# 다시 만든 분류기를 거치도록 합니다
test_sentence = '난 수업이 마치면 메리랑 놀거야'

test_sent_features = {word.lower():
                      (word in word_tokenize(test_sentence.lower()))
                      for word in all_words}
test_sent_features
```

Out[55]:
```
{'지루해': False,
 '메리는': False,
 '이쁜': False,
 '수업이': True,
 '난': True,
 '메리가': False,
 '마치고': False,
 '고양이야': False,
 '좋아': False,
 '고양이도': False,
 '놀거야': True,
 '메리랑': True}
```

In [56]:
```
# 부정의 결과가 나왔습니다
classifier.classify(test_sent_features)
```

Out[56]:
```
'neg'
```

In [57]:
```
# 이번엔 형태소를 분석하고 적용하겠습니다.
# 형태소를 분석하는 tokenize() 함수를 작성합니다
def tokenize(doc):
    return ['/'.join(t) for t in pos_tagger.pos(doc, norm=True, stem=True)]
```

In [58]:
```
# tokenize() 함수를 사용해서 연습용 데이터를 형태소 분석했습니다.
# 분석할 때는 태그를 붙여주는 것이 좋습니다.
train_docs = [(tokenize(row[0]), row[1]) for row in train]
train_docs
```

Out[58]:
```
[(['메리/Noun', '가/Josa', '좋다/Adjective'], 'pos'),
 (['고양이/Noun', '도/Josa', '좋다/Adjective'], 'pos'),
 (['난/Noun', '수업/Noun', '이/Josa', '지루하다/Adjective'], 'neg'),
 (['메리/Noun', '는/Josa', '이쁘다/Adjective', '고양이/Noun', '야/Josa'], 'pos'),
 (['난/Noun', '마치/Noun', '고/Josa', '메리/Noun', '랑/Josa', '놀다/Verb'], 'pos')]
```

In [59]:
```
# 연습용 데이터에 대해서
# 전체 말뭉치를 제작했습니다.
tokens = [t
          for d in train_docs
          for t in d[0]]
tokens
```

Out[59]:
```
['메리/Noun',
 '가/Josa',
 '좋다/Adjective',
 '고양이/Noun',
 '도/Josa',
 '좋다/Adjective',
 '난/Noun',
 '수업/Noun',
 '이/Josa',
 '지루하다/Adjective',
 '메리/Noun',
 '는/Josa',
 '이쁘다/Adjective',
```

In [60]:
```
# 말뭉치에 있는 단어가 있는지 없는지를
# 구분하는 함수를 정의합니다.
def term_exists(doc):
    return {word: (word in set(doc))
            for word in tokens}
```

In [61]:
```
# 연습용 데이터를 분석합니다.
# 명사와 조사의 구분이 잘 되어 있습니다.
train_xy = [(term_exists(d), c)
            for d,c in train_docs]
train_xy
```

Out[61]:
```
[({'메리/Noun': True,
   '가/Josa': True,
   '좋다/Adjective': True,
   '고양이/Noun': False,
   '도/Josa': False,
   '난/Noun': False,
   '수업/Noun': False,
   '이/Josa': False,
   '지루하다/Adjective': False,
   '는/Josa': False,
   '이쁘다/Adjective': False,
   '야/Josa': False,
   '마치/Noun': False,
   '고/Josa': False,
   '랑/Josa': False,
   '놀다/Verb': False},
  'pos'),
 ({'메리/Noun': False,
   '가/Josa': False,
```

In [62]:
```
# 새로 만든 테스트 문장이 분류기를 거치도록 합니다
classifier = nltk.NaiveBayesClassifier.train(train_xy)
```

In [63]:
```
# 형태소 분석을 하지 않았을 때 부정된 문장이
# 다시 분류기를 거치면 어떻게 나오는지 확인하기 위해
# 문장을 재사용합니다.
test_sentence = [("난 수업이 마치면 메리랑 놀거야")]
```

In [64]:
```
# 문장을 형태소 분석했습니다.
test_docs = pos_tagger.pos(test_sentence[0])
test_docs
```

Out[64]:
```
[('난', 'Noun'),
 ('수업', 'Noun'),
 ('이', 'Josa'),
 ('마치', 'Noun'),
 ('면', 'Josa'),
 ('메리', 'Noun'),
 ('랑', 'Josa'),
 ('놀거야', 'Verb')]
```

In [65]:
```
# 연습용 데이터에 붙어있는 긍정/부정 태그를 이용해서 분류했습니다.
classifier.show_most_informative_features()
```

```
Most Informative Features
                  난/Noun = True              neg : pos    =      2.5 : 1.0
                메리/Noun = False             neg : pos    =      2.5 : 1.0
           좋다/Adjective = False             neg : pos    =      1.5 : 1.0
              고양이/Noun = False             neg : pos    =      1.5 : 1.0
                 랑/Josa = False             neg : pos    =      1.1 : 1.0
                 는/Josa = False             neg : pos    =      1.1 : 1.0
                 가/Josa = False             neg : pos    =      1.1 : 1.0
               마치/Noun = False             neg : pos    =      1.1 : 1.0
                 야/Josa = False             neg : pos    =      1.1 : 1.0
               놀다/Verb = False             neg : pos    =      1.1 : 1.0
```

In [66]:
```
# 형태소 분석을 하지 않았을 때 부정된 문장이
# 다시 분류기를 거치도록 했습니다.
test_sent_features = {word: (word in tokens)
                      for word in test_docs}
test_sent_features
```

Out[66]:
```
{('난', 'Noun'): False,
 ('수업', 'Noun'): False,
 ('이', 'Josa'): False,
 ('마치', 'Noun'): False,
 ('면', 'Josa'): False,
 ('메리', 'Noun'): False,
 ('랑', 'Josa'): False,
 ('놀거야', 'Verb'): False}
```

In [67]:
```
# 긍정의 결과가 나왔습니다
classifier.classify(test_sent_features)
```

Out[67]:
```
'pos'
```

7. 문장의 유사도 측정하기

In [68]:
```
# 8-7 문장의 유사도 측정하기
# 8-5에서 실습한 Naive Bayes Classifier는
# 지도학습이기 때문에 미리 정답을 알고 있어야 합니다.
# 비지도학습, 즉 많은 문장이나 문서들 중에서
# 유사한 문장을 찾아내는 방법을 소개합니다.
```

In [69]:
```
# scikit-learn에서 텍스트의 특징을 추출하는 모듈에서
# CountVectorizer함수를 임포트 했습니다.
# 하나의 Counter Vector와 연습용 데이터를 생성합니다.
# CountVectorizer의 인자 min_df는
# 해당 값보다 빈도수가 낮은 단어를 무시하겠다는 뜻입니다.
# max_df 인자도 있는데, 이는 해당 값보다 많이 나오는,
# 빈도수가 높은 단어는 무시한다는 뜻입니다.
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df = 1)

contents = ['메리랑 놀러가고 싶지만 바쁜데 어떻하죠?',
                    '메리는 공원에서 산책하고 노는 것을 싫어해요',
                    '메리는 공원에서 노는 것도 싫어해요. 이상해요.',
                    '먼 곳으로 여행을 떠나고 싶은데 너무 바빠서 그러질 못하고 있어요']
```

In [70]:
```
# 연습용 데이터에 있는 단어들을
# 벡터화 한 것들을 feature로 잡습니다.
X = vectorizer.fit_transform(contents)
vectorizer.get_feature_names()
```

Out [70]:
```
['것도',
 '것을',
 '곳으로',
 '공원에서',
 '그러질',
 '너무',
 '노는',
 '놀러가고',
 '떠나고',
 '메리는',
 '메리랑',
 '못하고',
```

In [71]:
```
# 벡터 리스트값을
# 행렬로 출력합니다
X.toarray().transpose()
```

Out [71]:
```
array([[0, 0, 1, 0],
       [0, 1, 0, 0],
       [0, 0, 0, 1],
       [0, 1, 1, 0],
       [0, 0, 0, 1],
       [0, 0, 0, 1],
       [0, 1, 1, 0],
       [1, 0, 0, 0],
       [0, 0, 0, 1],
       [0, 1, 1, 0],
       [1, 0, 0, 0],
```

In [72]:
```
# 벡터의 크기를 출력합니다
X = vectorizer.fit_transform(contents)
num_samples, num_features = X.shape
num_samples, num_features
```

Out [72]:
```
(4, 22)
```

In [73]:
```
# 새로운 테스트 문장을 만들어
# 문장에 있는 단어들을 feature로 잡았습니다.
new_post = ['메리랑 공원에서 산책하고 놀고 싶어요']
new_post_vec = vectorizer.transform(new_post)
new_post_vec.toarray()
```

Out [73]:
```
array([[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]],
      dtype=int64)
```

In [74]:
```
# 한글 문장에 대한 벡터화
# Twitter 모델 엔진 임포트
from konlpy.tag import Twitter
t = Twitter()
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitter" ha
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
```

In [75]:
```
# 형태소 분석 결과를 contents_tokens에 저장합니다
# 출력해보면 '메리랑'과'메리는'을 '메리'로 분리해서
# 같은 단어로 출력되는 것을 볼 수 있습니다
contents_tokens = [t.morphs(row) for row in contents]
contents_tokens
```

Out[75]:
```
[['메리', '랑', '놀러', '가고', '싶지만', '바쁜데', '어떻하죠', '?'],
 ['메리', '는', '공원', '에서', '산책', '하고', '노', '는', '것', '을', '싫어해요'],
 ['메리', '는', '공원', '에서', '노', '는', '것', '도', '싫어해요', '.', '이상해요', '.'],
 ['먼',
  '곳',
  '으로',
  '여행',
  '을',
  '떠나고',
  '싶은데',
  '너무',
  '바빠서',
  '그러질',
  '못',
  '하고',
  '있어요']]
```

In [76]:
```
# 형태소 분석을 한 단어에 띄어쓰기로 구분한 것을
# 하나의 문장으로 만들어 scikit learn의
# vectorizer() 함수에서 사용하기 편하게 편집했습니다.
contents_for_vectorize = []

for content in contents_tokens:
    sentence = ''
    for word in content:
        sentence = sentence + ' ' + word

    contents_for_vectorize.append(sentence)

contents_for_vectorize
```

Out[76]:
```
[' 메리 랑 놀러 가고 싶지만 바쁜데 어떻하죠 ?',
 ' 메리 는 공원 에서 산책 하고 노 는 것 을 싫어해요',
 ' 메리 는 공원 에서 노 는 것 도 싫어해요 . 이상해요 .',
 ' 먼 곳 으로 여행 을 떠나고 싶은데 너무 바빠서 그러질 못 하고 있어요']
```

In [77]:
```
# feature을 찾았습니다
# 벡터화된 행렬의 크기를 출력합니다
X = vectorizer.fit_transform(contents_for_vectorize)
num_samples, num_features = X.shape
num_samples, num_features
```

Out[77]:
```
(4, 20)
```

In [78]:
```
# 벡터화된 행렬의 성분,
# 즉, feature를 확인했습니다.
vectorizer.get_feature_names()
```

Out[78]:
```
['가고',
 '공원',
 '그러질',
 '너무',
 '놀러',
 '떠나고',
 '메리',
 '바빠서',
 '바쁜데',
 '산책',
 '싫어해요',
```

In [79]:
```
# feature의 리스트를 받아
# 벡터화했습니다
X.toarray().transpose()
```

Out[79]:
```
array([[1, 0, 0, 0],
       [0, 1, 1, 0],
       [0, 0, 0, 1],
       [0, 0, 0, 1],
       [1, 0, 0, 0],
       [0, 0, 0, 1],
       [1, 1, 1, 0],
       [0, 0, 0, 1],
       [1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 1, 1, 0],
       [0, 0, 0, 1],
       [1, 0, 0, 0],
```

In [80]:
```
# 새로운 테스트 문장을 만들어 동일한 과정으로
# 벡터화해서 각 벡터들 사이의 거리를 구합니다
new_post = ['메리랑 공원에서 산책하고 놀고 싶어요']
new_post_tokens = [t.morphs(row) for row in new_post]

new_post_for_vectorize = []

for content in new_post_tokens:
    sentence = ''
    for word in content:
        sentence = sentence + ' ' + word

    new_post_for_vectorize.append(sentence)

new_post_for_vectorize
```

Out[80]:
```
[' 메리 랑 공원 에서 산책 하고 놀고 싶어요']
```

In [81]:
```
# feature을 찾아서 feature의 리스트를 받아 벡터화 했습니다.
new_post_vec = vectorizer.transform(new_post_for_vectorize)
new_post_vec.toarray()
```

Out[81]:
```
array([[0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]],
      dtype=int64)
```

In [82]:
```
# 새로운 문장과 비교해야 할 문장들
# 각각에 대해 거리를 구하기 위해
# 두 벡터의 차를 구하고 난 결과의
# norm을 구하는 함수를 정의합니다
import scipy as sp

def dist_raw(v1, v2):
    delta = v1 - v2
    return sp.linalg.norm(delta.toarray())
```

In [83]:
```
# 문장과 문장의 거리를 구했습니다
best_doc = None
best_dist = 65535
best_i = None

for i in range(0, num_samples):
    post_vec = X.getrow(i)
    d = dist_raw(post_vec, new_post_vec)

    print("== Post %i with dist=%.2f   : %s" %(i,d,contents[i]))

    if d<best_dist:
        best_dist = d
        best_i = i
```

```
== Post 0 with dist=3.00   : 메리랑 놀러가고 싶지만 바쁜데 어떻하죠?
== Post 1 with dist=1.00   : 메리는 공원에서 산책하고 노는 것을 싫어해요
== Post 2 with dist=2.00   : 메리는 공원에서 노는 것도 싫어해요. 이상해요.
== Post 3 with dist=3.46   : 먼 곳으로 여행을 떠나고 싶은데 너무 바빠서 그러질 못하고 있어요
```

In [84]:
```
# 문장의 거리가 가장 가까운 문장을 구했습니다.
# 즉, 문장의 의미가 반대지만 가장 흡사한 문장을 찾았습니다
print("Best post is %i, dist = %.2f" % (best_i, best_dist))
print('-->', new_post)
print('---->', contents[best_i])
```

```
Best post is 1, dist = 1.00
--> ['메리랑 공원에서 산책하고 놀고 싶어요']
----> 메리는 공원에서 산책하고 노는 것을 싫어해요
```

In [85]:
```
# 소속된 단어들의 조합을 확인했습니다
# 4개의 content 변수에 저장된 문장과
# 새로운 문장이 형태소 분석 후
# 벡터화된 결과를 확인했습니다
for i in range(0,len(contents)):
    print(X.getrow(i).toarray())

print('---------------------')
print(new_post_vec.toarray())
```

```
[[1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0]]
[[0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1]]
[[0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0]]
[[0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 1 1]]
---------------------
[[0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1]]
```

In [86]:
```
# 각 벡터의 norm을 나눠주고 거리를 구하도록 하는
# dist_norm함수를 정의했습니다
def dist_norm(v1, v2):
    v1_normalized = v1 / sp.linalg.norm(v1.toarray())
    v2_normalized = v2 / sp.linalg.norm(v2.toarray())

    delta = v1_normalized - v2_normalized

    return sp.linalg.norm(delta.toarray())
```

In [87]:
```
# 거리를 구한 결과가 조금 달라졌지만
# 가장 가까운 문장을 찾기에는 결과가 다르지 않았습니다.
best_doc = None
best_dist = 65535
best_i = None

for i in range(0, num_samples):
    post_vec = X.getrow(i)
    d = dist_norm(post_vec, new_post_vec)

    print("== Post %i with dist=%.2f    : %s" %(i,d,contents[i]))

    if d<best_dist:
        best_dist = d
        best_i = i
```

```
== Post 0 with dist=1.28   : 메리랑 놀러가고 싶지만 바쁜데 어떻하죠?
== Post 1 with dist=0.42   : 메리는 공원에서 산책하고 노는 것을 싫어해요
== Post 2 with dist=0.89   : 메리는 공원에서 노는 것도 싫어해요. 이상해요.
== Post 3 with dist=1.30   : 먼 곳으로 여행을 떠나고 싶은데 너무 바빠서 그러질 못하고 있어요
```

In [88]:
```
# 문장의 거리가 가장 가까운 문장을 구했습니다.
# 즉, 문장의 의미가 반대지만 가장 흡사한 문장을 찾았습니다
print("Best post is %i, dist = %.2f" % (best_i, best_dist))
print('-->', new_post)
print('---->', contents[best_i])
```

```
Best post is 1, dist = 0.42
--> ['메리랑 공원에서 산책하고 놀고 싶어요']
----> 메리는 공원에서 산책하고 노는 것을 싫어해요
```

In [89]:
```
# tf와 idf는 텍스트 마이닝에서 사용하는 일종의 단어별로 부과하는 가중치입니다
# tf는 어떤 단어가 문서 내에서 자주 등장할수록 중요도가 높을 것으로 봅니다
# idf는 비교하는 모든 문서에 만약 같은 단어가 있다면 이 단어가 핵심 어휘일지는
#       모르지만 문서간의 비교에서는 중요한 단어가 아니라는 뜻으로 봅니다
# tf와 idf의 원리를 적용하는 tfidf함수를 정의했습니다
def tfidf(t, d, D):
    tf = float(d.count(t)) / sum(d.count(w) for w in set(d))
    idf = sp.log( float(len(D))/(len([doc for doc in D if t in doc])) )
    return tf, idf
```

In [90]:
```
# tfidf함수에 새로 만든 연습용 데이터를 넣었을 때
# 결과를 출력했습니다
# 첫번째 출력에서 모든 문장에 a가 있으므로
# tf는 1이고, idf는 0입니다
a, abb, abc = ['a'], ['a','b','b'], ['a','b','c']
D = [a,abb,abc]

print(tfidf('a', a, D))
print(tfidf('b', abb, D))
print(tfidf('a', abc, D))
print(tfidf('b', abc, D))
print(tfidf('c', abc, D))
```

```
(1.0, 0.0)
(0.6666666666666666, 0.4054651081081644)
(0.3333333333333333, 0.0)
(0.3333333333333333, 0.4054651081081644)
(0.3333333333333333, 1.0986122886681098)
```

In [91]:
```
# tf와 idf값을 곱하도록 tfidf함수를 재정의합니다
def tfidf(t, d, D):
    tf = float(d.count(t)) / sum(d.count(w) for w in set(d))
    idf = sp.log( float(len(D))/(len([doc for doc in D if t in doc])) )
    return tf * idf
```

In [92]:
```
# 재정의한 tfidf() 함수 대신
# scikit-learn의 TfidfVectorizer를 임포트 해서 사용합니다
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=1, decode_error='ignore')
```

In [93]:
```
# 형태소 분석 결과를 contents_tokens에 저장했습니다
# 형태소 분석을 한 단어에 띄어쓰기로 구분한 것을
# 하나의 문장으로 만들어 scikit learn의
# vectorizer() 함수에서 사용하기 편하게 편집한다음,
# feature을 찾고 벡터화된 행렬의 크기를 출력합니다
contents_tokens = [t.morphs(row) for row in contents]

contents_for_vectorize = []

for content in contents_tokens:
    sentence = ''
    for word in content:
        sentence = sentence + ' ' + word

    contents_for_vectorize.append(sentence)

X = vectorizer.fit_transform(contents_for_vectorize)
num_samples, num_features = X.shape
num_samples, num_features
```

Out[93]: (4, 20)

In [94]:
```
# 만들어진 말뭉치를 확인했습니다.
vectorizer.get_feature_names()
```

Out[94]:
```
['가고',
 '공원',
 '그러질',
 '너무',
 '놀러',
 '떠나고',
 '메리',
 '바빠서',
 '바쁜데',
 '산책',
 '싫어해요',
 '싶은데',
 '싶지만',
 '어떻하죠',
 '에서',
 '여행',
 '으로',
 '이상해요',
 '있어요',
 '하고']
```

In [95]:
```
# 계속 사용하고 있는 테스트용 문장을 비교했습니다
new_post = ['근처 공원에 메리랑 놀러가고 싶네요.']
new_post_tokens = [t.morphs(row) for row in new_post]

new_post_for_vectorize = []

for content in new_post_tokens:
    sentence = ''
    for word in content:
        sentence = sentence + ' ' + word

    new_post_for_vectorize.append(sentence)

new_post_for_vectorize
```

Out[95]:
```
[' 근처 공원 에 메리 랑 놀러 가고 싶네요 .']
```

In [96]:
```
# 테스트 문장에 있는 단어들을 feature로 잡고,
# 다른 문장들과 거리를 비교했습니다
# 테스트 문장 '근처 공원에 메리랑 놀러가고 싶네요'과
# 가장 가까운 거리를 가지는 문장이
# '메리랑 놀러가고 싶지만 바쁜데 어떡하죠?'로 출력됨을
# 확인했습니다
new_post_vec = vectorizer.transform(new_post_for_vectorize)

best_doc = None
best_dist = 65535
best_i = None

for i in range(0, num_samples):
    post_vec = X.getrow(i)
    d = dist_norm(post_vec, new_post_vec)

    print("== Post %i with dist=%.2f    : %s" %(i,d,contents[i]))

    if d<best_dist:
        best_dist = d
        best_i = i

print("Best post is %i, dist = %.2f" % (best_i, best_dist))
print('-->', new_post)
print('---->', contents[best_i])
```

```
== Post 0 with dist=0.90    : 메리랑 놀러가고 싶지만 바쁜데 어떻하죠?
== Post 1 with dist=1.18    : 메리는 공원에서 산책하고 노는 것을 싫어해요
== Post 2 with dist=1.16    : 메리는 공원에서 노는 것도 싫어해요. 이상해요.
== Post 3 with dist=1.41    : 먼 곳으로 여행을 떠나고 싶은데 너무 바빠서 그러질 못하고 있어요
Best post is 0, dist = 0.90
--> ['근처 공원에 메리랑 놀러가고 싶네요.']
----> 메리랑 놀러가고 싶지만 바쁜데 어떻하죠?
```

8. 여자친구 선물 고르기

In [97]:
```
# 8-8 여자친구 선물 고르기
```

In [98]:
```
# 필요한 모듈을 임포트 하고 한글 폰트를 설정합니다
import pandas as pd
import numpy as np

import platform
import matplotlib.pyplot as plt

%matplotlib inline

path = "c:/Windows/Fonts/malgun.ttf"
from matplotlib import font_manager, rc
if platform.system() == 'Darwin':
    rc('font', family='AppleGothic')
elif platform.system() == 'Windows':
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
else:
    print('Unknown system... sorry~~~~')

plt.rcParams['axes.unicode_minus'] = False

from bs4 import BeautifulSoup
from urllib.request import urlopen
import urllib
import time
```

In [99]:
```
# 네이버 지식인에서 여자친구 선물을 검색해서
# 크롬 개발자 도구로 질문에 대한 답변이 위치한 곳의 태그가 dl임을 찾았습니다
# 주소창에 있는 주소 부분을 접근해야 할 주소를 정해 편집했습니다
tmp1 = 'https://search.naver.com/search.naver?where=kin'
html = tmp1 + '&sm=tab_jum&ie=utf8&query={key_word}&start={num}'

response = urlopen(html.format(num=1, key_word=urllib.parse.quote('여친 선물')))

soup = BeautifulSoup(response, "html.parser")

tmp = soup.find_all('dl')
```

In [100]:
```
# 한 페이지에 대해 테스트 결과를 출력합니다
tmp_list = []
for line in tmp:
    tmp_list.append(line.text)

tmp_list
```

Out[100]:
```
[' 기본검색  입력한 단어가 하나이상 포함된 문서 검색  ',
 ' 상세검색  정확히 일치하는 단어/문장(" ") 반드시 포함하는 단어(+) 제외하는 단어(-)
여러개의 단어를 입력하실 때는쉼표(,)로 구분해서 입력하세요.  ',
 ' 연관검색어도움말  파니니백  반지갑  웨스트아일랜드  질스튜어트  주얼리 브랜
유지됩니다. 연관검색어를 다시 보시겠습니까? 열기 ',
 '  질문  여친선물로 생로랑 틴트 펑이 좋아보이  2019.08.30.  여친선물로 생로랑 틴트
사줄려구 하는데요. 사는지역이 구미인데 구미에 구매 할수있는곳이 있나요? 아니면 대구
단 다른 오프매장은 마땅히 구매할 만한 곳이 없구요! 또 유통채널들 중에서 가장 다양하게
',
```

In [101]:
```
# 여자친구 선물에 대해 대략 1000개 정도의 검색 결과를 읽어왔습니다
from tqdm import tqdm_notebook

present_candi_text = []

for n in tqdm_notebook(range(1, 1000, 10)):
    response = urlopen(html.format(num=n, key_word=urllib.parse.quote('여자 친구 선물')))

    soup = BeautifulSoup(response, "html.parser")

    tmp = soup.find_all('dl')

    for line in tmp:
        present_candi_text.append(line.text)

    time.sleep(0.5)
```

100% 100/100 [01:17<00:00, 1.29it/s]

In [102]:
```
# 아주 많은 문장이 저장되어 있습니다
present_candi_text
```

Out[102]:
```
[' 기본검색  입력한 단어가 하나이상 포함된 문서 검색 ',
 ' 상세검색  정확히 일치하는 단어/문장(" ") 반드시 포함하는 단어(+) 제외하는 단어(-)  기본검
여러개의 단어를 입력하실 때는쉼표(,)로 구분해서 입력하세요.  ',
 ' 연관검색어도움말  여자친구목걸이  스킨케어  여자스킨로션추천  여자30대선물  에센스
바이스  선물  여자친구생일선물  여자근력운동  달기 후 1주일간 유지됩니다. 연관검색어를
 ' 질문  20대 여자친구 선물로 어떤 걸 해줘야 할까요??  2019.06.10.  이번에는 좀 색다른 걸
로 해줄건 다... 이미 줄 건 다 줘봤습니다. 20대 여자친구 선물로 실용적인 거 있음 추천해주세요
립니다. 20대 여자친구 선물로 보편적인 것은 반지, 팔찌, 목걸이 등등의... 이상으로 20대 여자친
 ' 질문  여자친구 선물(20-30)대 괜찮은거 뭐 있나요?  2019.08.26.  안녕하세요 여자친구 선
여자친구 선물이였음 좋겠는데 가격은 큰 상관없지만 특별하고 간직할 만한 그런 여자친구 선물이
쯤 고민해 봤을 것... 노미네이션을여자친구 선물로 추천하는 이유는 여친과의... 좀 캐쥬얼하고
'
```

In [103]:
```
# 문자열을 출력합니다
len(present_candi_text)
```

Out[103]:
```
1300
```

In [104]:
```
# 텍스트에서 의미있는 정보의 분석과 추출을
# 도와주는 자연어 처리 패키지 nltk 임포트
# 한글 문장에 대한 벡터화 하기위해 Twitter 모델 엔진 임포트
# 1만개의 문장에서 형태소 분석을 마친 단어를
# token_ko에 저장했습니다
import nltk
from konlpy.tag import Twitter; t = Twitter()

present_text = ''

for each_line in present_candi_text[:10000]:
    present_text = present_text + each_line + '₩n'

tokens_ko = t.morphs(present_text)
tokens_ko
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩konlpy₩tag₩_okt.py:16
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
```

Out[104]:
```
['기본',
 '검색',
 '입력',
 '한',
 '단어',
 '가',
 '하나',
 '이상',
 '포함',
 '된',
 '문서',
 '검색',
 '상세',
 '검색',
 '정확히',
 '일치',
```

In [105]:
```
# token으로 모은 단어는 95000개이고
# 그 중, 중복된 단어를 제외하면 378개입니다
ko = nltk.Text(tokens_ko, name='여자 친구 선물')
print(len(ko.tokens))
print(len(set(ko.tokens)))
```

```
94967
392
```

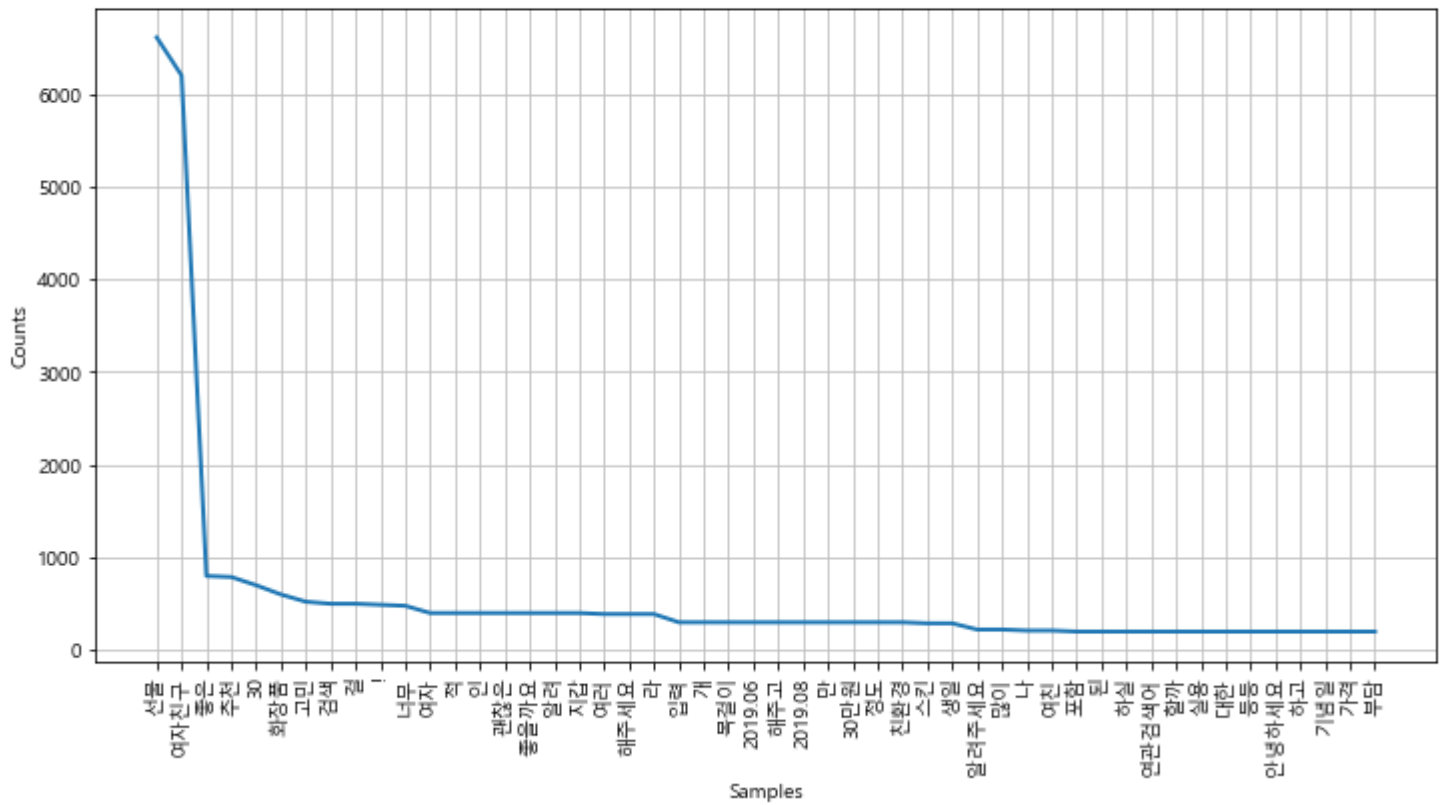
In [106]:
```
# 가장 많이 사용한 단어를 출력합니다
ko = nltk.Text(tokens_ko, name='여자 친구 선물')
ko.vocab().most_common(100)
```

Out[106]:
```
[('선물', 6611),
 ('여자친구', 6200),
 ('대', 3800),
 ('.', 3489),
 ('로', 3300),
 ('...', 2711),
 ('20', 2466),
 (',', 1489),
 ('거', 1400),
 ('답변', 1400),
 ('질문', 1100),
 ('?', 889),
 ('좋은', 800),
 ('추천', 789),
 ('이', 789),
 ('30', 700),
 ('가', 689),
 ('10', 634),
 ('것', 611),
```

In [107]:
```
# 중복된 단어를 제외해도 의미없는 단어들이 있어
# 다시 의미없는 단어를 제거했습니다
stop_words = ['.','가','요','답변','...','을','수','에','질문','제','를','이','도',
                        '좋','1','는','로','으로','2','것','은','다',',','니다','대','들',
                        '2017','들','데','..','의','때','겠','고','게','네요','한','일','할',
                        '10','?','하는','06','주','려고','인데','거','좀','는데','~','ㅎㅎ',
                        '하나','이상','20','뭐','까','있는','잘','습니다','다면','했','주려',
                        '지','있','못','후','중','줄','6','과','어떤','기본','!!',
                        '단어','선물해','라고','중요한','합','가요','....','보이','네','무지']

tokens_ko = [each_word
             for each_word in tokens_ko
             if each_word not in stop_words]

ko = nltk.Text(tokens_ko, name='여자 친구 선물')
ko.vocab().most_common(50)
```

```
('지갑', 400),
('여러', 389),
('해주세요', 389),
('라', 389),
('입력', 300),
('개', 300),
('목걸이', 300),
('2019.06', 300),
('해주고', 300),
('2019.08', 300),
('만', 300),
('30만원', 300),
('정도', 300),
('친환경', 300),
('스킨', 289),
('생일', 289),
('알려주세요', 222),
('많이', 222),
('나', 211),
```

In [108]:
```
# 단어의 등장 빈도에 따른 빈도수 그래프를 그렸습니다
plt.figure(figsize=(12,6))
ko.plot(50)
plt.show()
```



In [109]:

```
# 워드 클라우드를 그리기 위해 관련 모듈을 임포트하고
# 일반적인 워드 클라우드를 그렸습니다
from wordcloud import WordCloud, STOPWORDS
from PIL import Image

data = ko.vocab().most_common(300)

# for win : font_path='c:/Windows/Fonts/malgun.ttf'
wordcloud = WordCloud(font_path='/Library/Fonts/malgun.ttf',
                      relative_scaling = 0.2,
                      #stopwords=STOPWORDS,
                      background_color='white',
                      ).generate_from_frequencies(dict(data))
plt.figure(figsize=(10,6))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

In [110]:
```
# 하트 그림으로 마스킹하려고 합니다
mask = np.array(Image.open('../data/09. heart.jpg'))

from wordcloud import ImageColorGenerator
image_colors = ImageColorGenerator(mask)
```

In [111]:
```
# 지금까지 학습한 워드 클라우드를 출력합니다.
# malgun.ttf : 윈도우 10에서 적용
# AppleGothic.ttf : 맥OS에서 적용
# 출력하기 전에 윈도우 한글 폰트로 바꿔주었습니다.
data = ko.vocab().most_common(200)

# for win : font_path='c:/Windows/Fonts/malgun.ttf'
wordcloud = WordCloud(font_path='/Library/Fonts/malgun.ttf',
                      relative_scaling = 0.1, mask=mask,
                      background_color = 'white',
                      min_font_size=1,
                      max_font_size=100).generate_from_frequencies(dict(data))

default_colors = wordcloud.to_array()
```

In [112]:
```
# 하트 그림으로 그렸습니다
plt.figure(figsize=(8,8))
plt.imshow(wordcloud.recolor(color_func=image_colors), interpolation='bilinear')
plt.axis('off')
plt.show()
```

In [113]:
```
# 자연어 처리에서 word2vec을 지원하는
# gensim을 임포트합니다
import gensim
from gensim.models import word2vec
```

In [114]:
```
# 위에서 얻은 샘플에 대해
# 조사와 어미 등을 제거하는 과정입니다
twitter = Twitter()
results = []
lines = present_candi_text

for line in lines:
    malist = twitter.pos(line, norm=True, stem=True)
    r= []

    for word in malist:
        if not word[1] in ["Josa", "Eomi", "Punctuation"]:
            r.append(word[0])

    r1 = (" ".join(r)).strip()
    results.append(r1)
    print(r1)
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩konlpy₩tag₩_okt.py:16: UserWarning: "Twitter" has changed to "Okt" since KoNLPy v0.4.5.
  warn('"Twitter" has changed to "Okt" since KoNLPy v0.4.5.')
```

기본 검색 입력 단어 하나 이상 포함 되다 문서 검색
상세 검색 정확하다 일치 하다 단어 문장 반드시 포함 하다 단어 제외 하다 단어 기 본 검색 결과 범위 좁이다 하다 때 사용 하다 여러 개 단어 입력 하다 때 쉼표 로 구분 하다 입력 하다
연관검색어 도움말 여자친구 목걸이 스킨 케어 여자 스킨로션 추천 여자 30 대 선물 에센스 추천 20 대 여자친구 선물 여자친구 선물 뷰티 디바이스 선물 여자친구 생일 선물 여자 근력 운동 달다 후 1 주 일간 유지 되다 연관검색어 다시 보다 열기
질문 20 대다 여자친구 선물 어떻다 걸 해주다 하다 요 2019.06 10 이번 좀 색다르다 걸 해주다 싶다 웬만하다 20 대다 여자친구 선물 해주다 다 이미 줄 건 다 주다 보다 20 대다 여자친구 선물 실용 적 거 있다 추천 해주다 답변 20 대다 여자친구 선물 대한 답변 드리다 20 대다 여자친구 선물 보편 적 것 반지 팔찌 목걸이 등등 이상 20 대다 여자친구 선물 대한 답변 이다
질문 여자친구 선물 20-30 대다 괜찮다 거 뭐 있다 2019.08 26 안녕하다 여자친구 선물 고민 있다 늘 기념일 싶다 여자친구 선물 이다 좋다 가격 크다 상관없다 특별하다 간 직할 만 그렇다 여자친구 선물 이면 답변 여자친구 선물 이 것 누구 한번 쯤 고민 하다 보다 것 노미 네이션 여자친구 선물 추천 하다 이유 여친 좀 캐 쥬얼 부담 없다 가격 대 여자친구 선물 중
질문 30 대 여자친구 선물 여러 개 추천 좀 해주다 2019.08 22 30 대 여자친구 선물 좋다 거 여러 개 추천 해주다 그 중 골 선물 하다 그동안 30 대 여자친구 선물 함 향수 시계 목걸이 등등 딱하다 엄청나다 좋아하다 답변 30 대 여자친구 선물 같다 경우 실용 적 걸 원하다 경우 많다 그간 사주다 답변 좀 더 건강 깊숙하다 관련 되다 30 대 여자친구 선물 이 더 좋아하다 같다
질문 20 대 여자친구 선물 뭘 주다 좋아하다 요 2019.08 13 20 대 여자친구 선물 검색 하다 다 거기 거기 선물 들 뜨다 여기 질문 남기다 보다 새롭다 걸 선물 해주다 싶다 20 대 여자친구 선물 어떻다 좋다 답변 안녕하다 20 대 여자친구 선물 어떤 것 준비 하다 고민 하다 제 좋다 것 알다 드리다 20 대 여자친구 선물 준비 하다 너무 좋아하다 좋다 걸 알다 드리다

In [115]:
```
# 수정한 데이터를 저장했습니다
data_file = 'pres_girl.data'
with open(data_file, 'w', encoding='utf-8') as fp:
    fp.write("₩n".join(results))
```

In [117]:
```
# 저장한 데이터를 읽어왔습니다
# 선물과 유사한 단어를 출력했습니다
model = word2vec.Word2Vec.load("pres_girl.model")
model.most_similar(positive=['선물'])
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩ipykernel_lau
removed in 4.0.0, use self.wv.most_similar() instead).
  after removing the cwd from sys.path.
```

Out[117]:
```
[('여자친구', 0.8186136484146118),
 ('좋다', 0.4012829661369324),
 ('20', 0.3802419304847717 3),
 ('거', 0.35115671157836914),
 ('해주다', 0.3140711486339569),
 ('질문', 0.3038713335990906),
 ('것', 0.30117642879486084),
 ('만', 0.2935858368873596),
 ('스킨', 0.2926303744316101),
 ('고민', 0.2898249328136444)]
```

In [118]:
```
# 여자친구와 유사한 단어를 출력했습니다
model.most_similar(positive=['여자친구'])
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩ipykernel_lau
removed in 4.0.0, use self.wv.most_similar() instead).
```

Out[118]:
```
[('선물', 0.8186136484146118),
 ('좋다', 0.37473610043525696),
 ('거', 0.35325294733047485),
 ('20', 0.3504440188407898),
 ('스킨', 0.34262561798095703),
 ('것', 0.2985553443431854),
 ('고민', 0.29694128036499023),
 ('노아', 0.2900157570838928),
 ('만', 0.28386008739471436),
 ('답변', 0.2815096974372864)]
```

In [119]:
```
model.most_similar(positive=['스킨', '목걸이'])
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.
removed in 4.0.0, use self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
```

Out[119]:
```
[('케어', 0.6829251050949097),
 ('스킨로션', 0.6536237597465515),
 ('도움말', 0.6138361692428589),
 ('등등', 0.5950197577476501),
 ('모로', 0.5811956524848938),
 ('해보다', 0.5696347951889038),
 ('에센스', 0.5661535263061523),
 ('로션', 0.565956592559814 5),
 ('뷰티', 0.5613692998886108),
 ('엄청나다', 0.4884209632873535)]
```

In [120]:
```
model.most_similar(positive=['여자친구'], negative=['시계'])
```

```
C:₩Users₩nerin₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.
removed in 4.0.0, use self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
```

Out[120]:
```
[('선물', 0.5623495578765869),
 ('20', 0.2466636598110199),
 ('보다', 0.2448594868183136),
 ('만', 0.23310431838035583),
 ('크다', 0.2174287587404251),
 ('좋다', 0.20567268133163452),
 ('늘', 0.1995995044708252),
 ('탁월하다', 0.18321172893047333),
 ('고민', 0.18306584656238556),
 ('것', 0.17761966586112976)]
```

9. 소감

이번 과제에서는 자연어 처리를 어떻게 할 수 있는지 알아보았습니다.

재미있었던 점은 예전에 그림에 글자가 채워져 있는 그림을 보고 저걸 어떻게 하나하나 다 채워넣었나 하고 생각했었는데
그것들은 사람이 직접 채운게 아니라 워드 클라우드라는 기능을 사용하면 쉽게 만들 수 있었다는걸 알게 되었습니다.

어려웠던 점은 8-7 CountVecctorizer 함수를 사용할 때
문장을 단어로 나눠 벡터화시킨것을 feature로 잡는다? 벡터 리스트 값을 행렬로 출력한다? 등
벡터의 개념이 잘 이해가 되지 않았습니다.
이 부분에 대해선 기말 프로젝트를 진행하면서 필요하다면 조금 더 알아보도록 하겠습니다.